1 Reactive Systems and Temporal Properties

1.1 Example: the alternating bit protocol

- Channels may drop (or perhaps duplicate).
- Sender retransmits (at some interval) until matching ack received.
- Sequence numbers prevent duplication of msgs or acks.
- Sequence numbers are modulo 2 (hence, “alternating bit”).

This is an example of a “reactive” system [3] (Pnueli):

- “Reacts” to stimulus from environment.
- Does not terminate.

Note each component (sender, receiver, channels) is also a reactive system.

1.2 Temporal properties

To reason about reactive systems and the interaction of their components, we need to be able to state *temporal properties*.

E.g., for the alternating bit protocol:
• Every message sent is eventually received.
• A message is not received unless one is sent
• If \( x \) is sent before \( y \), then \( x \) is received before \( y \).

Some properties of the components:

• Sender continues to resend msg until ack.
• If channel continues to receive input, it eventually transmits (does not drop) a msg.
• Recvr does not produce ack before msg is output.
• \textit{etc.}

Note: these are properties about relationships in time \textit{(i.e., temporal properties)}.

### 1.3 Formalizing temporal properties

…to specify and reason about reactive systems.

• Consider using first order logic to write temporal properties, representing time by a natural number \( t \).

  For example: “every time an \( x \) msg is input, one is eventually output”

\[
\forall t \geq 0 : \text{input}(x, t) \Rightarrow \exists t' \geq t : \text{output}(x, t')
\]

  This is adequate, but a bit hard to read!

• Temporal Logic

  Pnueli suggested using \textit{temporal logic} to express properties of reactive systems. In temporal logic, the time parameter \( t \) is implicit:

  - \( G \ p \) true at time \( t \) if \( p \) is true at \textit{all} \( t' \geq t \).

    \[
    p \quad p \quad p \quad p \quad p \quad p \quad p \quad p \quad p \quad p \quad \ldots
    \]

    \[
    Gp\ldots
    \]
– \( F \ p \) true at time \( t \) if \( p \) is true at some \( t' \geq t \).

\[ \begin{array}{cccc}
\text{p} & \text{p} & \text{p} & \text{p} \\
\text{Fp} & ... & ... & \text{Fp}
\end{array} \]

Note, \( G \) and \( F \) are dual:

\[ Gp \equiv \neg F \neg p \]
\[ Fp \equiv \neg G \neg p \]

Here are, for example, some other equivalences:

\[ Gp \land Gq \equiv G(p \land q) \]
\[ Fp \lor Fp \equiv F(p \lor q) \]

But note,

\[ Gp \lor Gq \not\equiv G(p \lor q) \]
\[ Fp \land Fp \not\equiv F(p \land q) \]

Our previous example in temporal logic:

\[ G(\text{input}(x) \Rightarrow F \text{output}(x)) \]

This can be read “always, if input\((x)\) then eventuall output\((x)\).” It is an example of a liveness property, since it states some “good” condition that must eventually occur.

- “Infinitely often” properties
  - Note \( G \ F \ p \) means that \( p \) occurs infinitely often (“always eventually \( p \)”). This is equivalent by De Morgan’s laws to \( \neg F \ G \neg p \) or “a point is never reached where \( p \) is forever false”.
  - example:
    \[ GF\text{send}_{\text{on}_{\text{chan}}}(x) \Rightarrow GF\text{recv}_{\text{on}_{\text{chan}}}(x) \]
    “If msg \( x \) is sent infinitely often, it is received infinitely often”
    This is an example of a “fairness” property.

- The “until” operator
  \( p \ U q \) true at time \( t \) if
  - \( q \) is true at some \( t' \geq t \), and
  - \( p \) is true in the range \([t, t')\)
This can be read as “$q$ is eventually true, and until that time $p$ remains true.”

The “weak” until:

$$p\mathcal{W}q \equiv p \ U \ q \lor Gp$$

That is, the weak until allows the possibility that $q$ never happens and $p$ remains true forever. This is useful for expressing properties like:

$$\neg\text{output}(x) \ W \ \text{input}(x)$$

“An output does not occur before an input occurs.” This is an example of a safety property. It states some “bad” condition that should never occur.

Note: the formal distinction between safety and liveness is the following:

- A safety property, if false, can always be proved false by exhibiting a finite run.
- A liveness property can only be proved false by exhibiting an infinite run – any finite run can be extended so that it satisfies the eventuality condition.

• Some other temporal operators:

  - The “next time” operator: $Xp$ is true at time $t$ if $p$ is true at time $t + 1$.
  - Past time operators: $H$, $P$ and $\mathcal{S}$ are “past time” versions of $G$, $F$ and $\mathcal{U}$ respectively.
    Example:
    $$G(\text{output}(x) \Rightarrow P \text{ input}(x))$$
    expresses the same property as above “input must occur before output”.

1.4 Model theory for temporal logic

• To interpret temporal logic formulas formally, we take as our structure an infinite sequence of states

  $$\sigma = s_1, s_2, s_3, \ldots$$

  We write

  - $\sigma, s_i \models \phi$ if formula $\phi$ is true in state $s_i$ of sequence $\sigma$,
  - $\sigma \models \phi$ if $\phi$ is true in the first state of $\sigma$
  - $\models \sigma$ if $\sigma$ is valid (true in all models).
Propositional linear temporal logic (or PLTL) is a set of formulas defined as follows:

A formula in PLTL is either an atomic proposition, or one of the following:

$$\text{true, } p \lor q, \neg p, p U q, Xp$$

where $p$ and $q$ are formulas.

Note: An atomic proposition is simply a propositional letter which takes on the value true or false in any given state. For our purposes, formulas like “input($x$)” are atomic propositions.

The remaining operators of PLTL can be viewed as derived operators:

$$p \land q \equiv \neg(\neg p \lor \neg q)$$
$$Fp \equiv \text{true } U q$$
$$Gp \equiv \neg F \neg p$$

etc. . .

Definition of PLTL satisfaction:

$$\sigma, s_i \models a \quad (\text{an atomic prop}) \quad \text{iff} \quad s_i \models a$$
$$\sigma, s_i \models \neg p \quad \text{iff} \quad \sigma, s_i \not\models p$$
$$\sigma, s_i \models p \lor q \quad \text{iff} \quad \sigma, s_i \models p \text{ or } \sigma, s_i \models q$$
$$\sigma, s_i \models Xp \quad \text{iff} \quad \sigma, s_{i+1} \models p$$
$$\sigma, s_i \models p U q \quad \text{iff} \quad \text{for some } j \geq i, \sigma, s_j \models q$$
$$\quad \text{and for all } i \leq k < j, \sigma, s_k \models p$$

1.5 Proofs in temporal logic

Using a proof system for PLTL, we can, for example, formally prove some properties of the ABP, given some assumptions about its component parts (sender, channels and recvr).

- A proof is a sequence of formulas, each of which is either
  - An instance of an axiom, or
  - the result of applying an inference rule to earlier formulas in the sequence.

- The particular choice of axioms and inference rules for PLTL is not of much interest. It suffices to know that a set of such axioms and rules exists that is
  1. sound (no invalid formulas can be proved)
  2. complete (all valid formulas can be proved)
• Derived inference rules
  Of more interest are “derived” inference rules, which can be proved valid from the
  axioms and primitive inference rules, and are useful for program proofs. For example:

  – Chaining eventualities

    \[
    \begin{align*}
    G(p \Rightarrow Fq) \\
    G(q \Rightarrow Fr) \\
    \hline
    G(p \Rightarrow Fr)
    \end{align*}
    \]

  – Proving invariance

    \[
    \begin{align*}
    G(p \Rightarrow Xp) \\
    \hline
    Gp
    \end{align*}
    \]

• A (partial) proof of liveness for ABP

  We make the following assumptions about the system components:

  From these, we want to prove \(G(\text{input} \Rightarrow \text{Foutput})\). The following is a sketch of the proof:

  The following assumption:

  \(A1:\ G(\text{input} \land \text{send}_i \Rightarrow (F\text{ack}_\text{recv}_i \lor GF\text{msg_send}_i))\)

  states that the sender on receiving input, when the send count is \(i\), either eventually
  receives an ack (numbered \(i\)), or it retransmits a msg (numbered \(i\)) forever. We combine
  this with the fairness assumption

  \(A2:\ G(GF\text{msg_send}_i \Rightarrow GF\text{msg_recv}_i)\)

  on the message channel to infer

  \(G(\text{input} \Rightarrow (F\text{ack}_\text{recv}_i \lor GF\text{msg_recv}_i))\)

  This in turn is combined with the liveness property of the receiver:

  \(A3:\ G(\text{recv}_i \land F\text{msg_recv}_i \Rightarrow \text{Foutput}))\)
To obtain the following:

\[ G(\text{input} \land \text{send}_i \land \text{recv}_i \Rightarrow (\text{Ack}_{\text{rcv}_i} \lor \text{Foutput}) \]

That is, eventually, either an ack is received by the sender, or an output is produced (if the sender and receiver count are both \( i \) when the input arrives). We now need to prove two safety lemmas. The first states that when input arrives, the receiver count always matches the sender count:

\[
\text{safety lemma}_1 : \quad G(\text{input} \land \text{send}_i \Rightarrow \text{recv}_i)
\]

(proof omitted). From this we infer

\[ G(\text{input} \land \text{send}_i \Rightarrow (\text{Ack}_{\text{rcv}_i} \lor \text{Foutput})) \]

The second safety lemma states that a spurious ack is not generated before an output has been produced. We will take this lemma as a given:

\[
\text{safety lemma}_2 : \quad G(\text{input} \land \text{send}_i \Rightarrow (\neg \text{Ack}_{\text{rcv}_i} \lor \text{Woutput}))
\]

From this and the preceding, we infer

\[ G(\text{input} \land \text{send}_i \Rightarrow (\text{Foutput} \lor \text{Foutput})) \]

Since \( G(\text{send}_0 \lor \text{send}_1) \) holds, we have:

\[ G(\text{input} \Rightarrow \text{Foutput}) \]

Note:

- This is just a proof sketch – the details of each step have to be filled in (perhaps by treating each step as an instance of a derived rule).
- We need to prove safety properties to prove the liveness property. The proof of these lemmas is actually quite a bit more involved than the liveness proof. [Hailpern uses 15 pages to prove the ABP]
- The above proof sketch probably contains a fallacy.
- There has to be a better way!

## 2 Model Checking

(Clarke/Emerson, Queille/Sifakis)

Instead of the previous approach of proving temporal properties of the system from temporal properties of the components, we can take the model checking approach [1]:
1. Build a finite state (usually abstract) model $M$ of the protocol.

2. Check automatically that $M \models f$, where $f$ is a desired temporal property, or

3. produce a counterexample automatically if $M \not\models f$.

A Kripke Model $(S, R, L)$ consists of

1. a set of states $S$,
2. a set of transitions between states $R$,
3. a labeling $L$, giving the value of each atomic proposition in each state.

Example: Kripke model for a very simple sequential program:

```plaintext
repeat
  p := true;
  p := false;
end
```

2.1 Example: modeling a protocol in CSP (Hoare)

- A CSP program [2] consists of a collection of parallel processes with only local variables:

  ```plaintext
  program ::
  proc1 :: <local var decls> <statement>
  || proc2 :: <local var decls> <statement>
  ... || procn :: <local var decls> <statement>
  ```

- Simple statements
  
  - skip (do nothing)
  - $x := x + 1$ (local assignment)

- Sequential composition

  ```plaintext
  y := 1; x := x + y
  ```

- Communication by synchronous message passing
- P!y send value of y to process P
- Q?x receive value of x from process Q

Both send and receive actions are blocking. That is, a sender must wait until receiver is ready to receive and vice-versa.

- Nondeterministic choice operator: □
  
  sender?x; [recvr!x □ skip]
  
  (Choose nondeterministically to transmit message or do nothing)

- Guarded commands: conditions ⇒ statement
  
  chan?x; [x > last ⇒ output!x □
  x ≤ last ⇒ skip]
  
  (If the received number greater than last, then transmit, else do nothing)

- Iteration: [statement] ∗
  Terminates when all guards are false:
  
  chan?x; [last_ack < x ⇒
  chan!x □ ack_chan?last_ack]
  ∗
  
  (Continue sending until receive an ack ≥ x)

- Example – a mutual exclusion protocol
  Two processes must be prevented from entering their critical region simultaneously:

  p[i=1,2] :: [ T1 :: M?enter(); C1 :: skip; (abstracts critical region) Mexit();
  N :: skip; (abstracts non-critical section) Mtry();
  ] ∗

  They communicate with a process M that enforces mutual exclusion and guarantees eventual access:
M ::
try[1,2] : boolean, initially 0;
turn : 0..2, initially 0;

L ::
[ p[i=1,2]?try() ⇒
  [turn = 0 ⇒ turn := i];
  try[i] := 1
 □ i=1,2: try[i] ∧ (turn ≠ 3-i) ⇒
  p[i]!enter();
E_i ::
  p[i]?exit();
  try[i] := 0;
  [try[3 - i] ⇒ turn := 3 - i]
=* 

• Program state
  A state consists of
  — a program statement label for each process
  — a valuation for each local variable
  Example: \((\{T_1,C_2,E_2\}, \text{try}[1] = 1, \text{try}[2] = 1, \text{turn} = 2)\)

• Interleaving concurrency
  Concurrency is modeled by allowing at each state a nondeterministic choice of which
  process to advance. (This is like simulating concurrency by coroutines).
  Example: from the following program fragment:
  
  \[
  \begin{align*}
  A &:: x := x + 1; & B &:: \text{skip} \\ C &:: y := y + 1; & D &:: \text{skip}
  \end{align*}
  \]

  we obtain this state graph:
  \[
  \begin{array}{c}
  \{(A,C), x=0, y=0\} \\
  \{(B,C), x=1, y=1\} \\
  \{(A,D), x=0, y=1\} \\
  \{(B,D), x=1, y=1\}
  \end{array}
  \]

• Synchronized transitions
  A receive and a send combine to produce a single synchronized transition.
  Example:
A:: P!0; B:: skip ∥
C:: Q?x; D:: skip

produces (for example) the following transition:

\[
(A,C), x=1 \\
\downarrow \\
(B,D), x=0
\]

- **Example:** Generating a Kripke model from a CSP program

The mutual exclusion protocol:

1. Start with the initial state
2. generate all possible transitions from that state

\[
(N1,N2,L), \quad t[1]=0, t[2]=0, turn=0
\]

\[
(T1,N2,L), \quad t[1]=1, t[2]=0, turn=1
\]

\[
(N1,T2,L), \quad t[1]=0, t[2]=1, turn=2
\]

3. repeat on new states until none (breadth-first search)

Final result:

- Interpreting temporal formulas on Kripke models

A *path* in a Kripke model \( M = (S, R, L) \) is any infinite sequence

\[
\sigma = s_1, s_2, s_3, \ldots
\]

of states in \( S \) such that every pair \((s_i, s_{i+1})\) is a transition in \( R \).
If $F$ is a PLTL formula, we say
\[ M, s_1 \models f \]
when for every path $\sigma = s_1, s_2, s_3, \ldots, \sigma \models f$.
In our example, we have $M, s_{\text{init}} \models G(T_1 \Rightarrow F C_1)$, for example. To verify this automatically, we require another digression…

References

